Electronic Version
Stylesheet Version v1.1.1

Description

System, Method, and Computer Program Product of Building A Native XML Object Database

BACKGROUND OF INVENTION

FIELD OF THE INVENTION

[0001] The present invention is about building an NXOD - Native XML Object Database that is object oriented in the API layer and native XML in the persistence layer.

RELATED ART

[0002] A typical database system comprises of three layers: API, database core, and data persistence. The API layer takes orders from external database applications. The database core processes orders, causes physical changes in the persistence layer, reports the processing result to the API layer, which in turn reports to external applications.

[0003] Modern database technology renders mainly three cate-gories of databases: relational database, object database, and XML database. Relational databases focus on relationships between tables and integrity of data while object and XML databases emphasis on mapping real world entity relationships to the structure of data. XML databases use XML in either or both of the API and persistence layers.

[0004] Database applications interact with databases mainly through three categories of APIs: SQL, Get/Set operations, and SOAP. SQL is a query language for relational databases; Get/Set operations are programming interfaces for object databases, get operation being for data retrieval, set for data update/removal; SOAP is an XML plain text messaging mechanism for XML databases.

[0005] A data file at the persistence layer can take a proprietary binary form or a plain text form. Most of the relational and object databases use a binary form. XML databases use an XML plain text form or a binary form.

[0006] XML databases comprise of XML enabled database and Native XML database. A relational or object database is an XML enabled database if it supports SOAP API or XML data input/output. A native XML database uses XML for data file format at the persistence layer, or saves data to proprietary data stores via DOM - Document Object Model.

SQL Server 2000 by Microsoft, for instance, is an XML enabled relational database management system; Matisse by Matisse Software an XML enabled object database management system; dbXML by dbXML Group a native XML database management system.

[0007] The present invention uses Get/Set operations at the API layer and XML files for data persistence. That is how NXOD - Native XML Object Database gets named.

[0008] Database systems on the market support a variety of programming interfaces: C, C++, Java, Perl, etc. All of the APIs are static, manually coded, and shipped along with their respective products. The present invention provides means and steps of dynamically generating APIs based on object oriented design. Dynamic NXOD API is more user friendly, which cuts database application development time. Dynamic NXOD API also embeds data links to achieve fast query processing and database transactions.

[0009] In the persistence layer, US Patent Application #20040103105 by Christopher Lindablad and Paul Padersen proposes a tree like hierarchy for the data store. As it assumes static APIs, data links are embedded in the tree. Also, It does not examine data persistence in a coherent lifecycle of design, API, and storage. In XNOD, API and storage hierarchy are driven by object oriented design.

[0010] Existing native XML databases are trying to store the structure of data in XML files, which has performance penalties due to XML parsing of nested structures. The present invention separates structure and value of data. The hierarchy is represented by file system paths. Native XML files have flat structures and store name/value pairs. No elements except the document root in the XML file has child elements. When viewing the data store as a tree, all the inner nodes including the root are file directories; all leaf nodes are XML files.

[0011] At the database design phase, all three categories of databases start from entity relationship diagrams. Relational databases then map the entity relationship diagrams to tables that satisfy Boyce-Codd Normal Form; object and XML databases to a hierarchy of objects.

[0012] The present invention starts from entity relationship diagrams. The data representation, however, is both a set of relational tables and a hierarchy of objects.

SUMMARY OF INVENTION

[0013] The present invention provides steps and means for building an NXOD - Native XML Object Database that offers a combination of features from heterogeneous database systems. NXOD follows an object-oriented design. The data representation is both of normalized tables like in relational databases, and of a tree of objects like in object databases. At the front end, NXOD offers a set of getters and setters like in object databases. At the back-end, NXOD saves data in XML files like in native XML databases.

[0014] The present invention comprises of following differentiators:1) Dynamic design driven API, 2) Separation of structure and value of data, 3) Data links embedded in dynamic API, 4) Better granularity for data access control and encryption, 4) Better reliability and resilience to data corruptions, 5) Smaller memory footprint, faster query processing and operations.

[0015] In a word, being a hybrid of native XML and object databases, the present invention innovates in API, implementation, storage, security, and performance.

Brief Description of Drawings

[0016] *Fig. 1* depicts an *embodiment* of the present invention NXOD to interact with an external database application via Get/ Set operation pairs.

[0017] *Fig. 2* depicts a generic workflow of the present invention.

[0018] *Fig. 3* depicts a sample object oriented design utilized by NXOD.

[0019] *Fig. 4* depicts a sample mapping from said design to a file system structure.

[0020] *Fig. 5* depicts the database core and its internal working.

DETAILED DESCRIPTION

[0021] The present invention hides all the complexity of database queries from end users. As depicted in *Fig. 1*, NXOD *180* interacts with external applications *100* via Get/Set operations.

[0022] *Fig. 2* offers a close view of how the present invention works in real world. First, the Database Application *200* talks to dynamic API *240*, which talks to the Database Core *260*, which manipulates native XML files *280*.

DATA MAPPING

[0023] The present invention provides steps and means for transparently mapping an object oriented database design, an entity relationship diagram as depicted in *Fig. 3*, to the file system structure as depicted in *Fig. 4*. For simplicity, said database design comprises two entities: Primary Holder *310* and Bank Account *380*. The relationship is 1 to N, i.e., one primary account holder can have one or more bank accounts, but one bank account can only have one primary holder. Primary Holder has four attributes: SSN *312*, Account Numbers *314*, First Name *316*, and Last Name *318*. Account Numbers holds a list of bank account numbers that reference to Bank Account *380*.

[0024] Bank Account has three attributes: Account Number *382*, Bank Name *384*, and Balance*386*. Bank Account has two descendant entities: Checking Account *390* and Brokerage Account *396*. Checking account has an attribute Overdraft *392*; Brokerage Account an attribute Margin *398*.

[0025] The present invention follows said design and *dynamically* generates following interfaces and classes, which can be done in any object-oriented programming language. See the Program Listing Deposit for Java examples.

[0026] *1. Interfaces:* i) IPrimaryHolder extends IIdentity, ii) IBankAccount extends IIdentity, iii) ICheckingAccount extends IBankAccount, iv) IBrokerageAccount extends IBankAccount.

[0027] *2. Implementation Classes:* i) PrimaryHolder implements IPrimaryHolder, ii) BankAccount implements IBankAccount, iii) CheckingAccount extends BankAccount implements IBankAccount, iv) BrokerageAccount extends BankAccount implements IBrokerageAccount.

[0028] *3. Query Classes:* i) PrimaryHolders runs queries for Primary-Holder, ii) BankAccounts runs queries for BankAccount, iii) CheckingAccounts runs queries for CheckingAccount, iv) BrokerageAccounts runs queries for BrokerageAccount.

[0029] Running sample database *Application A* in the Program Listing Deposit stores Primary Holder data in the XML content:

[0030]
```
<?xml version="1.0"encoding="utf-8" ?>
<PrimaryHolder> <ssn>123</ssn>
<userName>a_user</userName>
<firstName>John</firstName>
<lastName>Smith</lastName>
<accountNumber>456</accountNumber>
<accountNumber>789</accountNumber>
</PrimaryHolder>
```

[0031] Running sample database *Application B* in the Program Listing Deposit stores Checking Account data in the XML content:

[0032]
```
<?xml version="1.0"encoding="utf-8" ?>
<CheckingAccount> <userName>a_user</userName>
<accountNumber>456</accountNumber>
<bankName>a_bank</bankName>
<balance>2000.68</balance>
```

<overdraft>1000.00</overdraft> </CheckingAccount>

[0033] Running sample database *Application C* in the Program Listing Deposit stores Brokerage Account data in an XML content:

[0034] <?xml version="1.0"encoding="utf-8" ?> <BrokerageAccount> <userName>a_user</userName> <accountNumber>456</accountNumber> <bankName>b_bank</bankName> <balance>8000.26</balance> <margin>yes</margin> </BrokerageAccount>

[0035] Said XML contents are saved in the file system as depicted in *Fig. 4*. The ROOT DIRECTORY *400* is created while NXOD is being loaded. Execution of said sample database applications causes four directories and three files to be created. Directory PrimaryHolders *420* contains XML file *426.* Directory BankAccount *440* has two sub directories (to map said inheritance of bank account entities): CheckingAccounts *60* and BrokerageAccounts *480.* CheckingAccounts contains XML file *466;* BrokerageAccounts XML file *486 .*

[0036] Each XML file stores one instance of the data object. Data for 1000 primary account holders will be stored in 1000 XML files under the PrimaryHolders directory *420.* Each primary account holder can have one or more checking accounts. The number of XML files under the CheckingAccounts directory *460* is the total number of checking accounts held by all primary account holders. The number of XML files under the BrokerageAccounts directory *486* is the total number of brokerage accounts held by all primary account holders.

[0037] The present invention separates structure from value of data. As depicted in *Fig. 4*, the hierarchy is represented by file paths. Flat name/value pairs are stored in XML files *426*, *466*, and *486*.

DATABASE QUERY

[0038] The present invention eliminates the need for a query language like SQL or XQuery. An NXOD query is initiated by the external application *200* and executed by a series of get operations in the API *240* and the database core *260*.

[0039] Locating an instance of data is by following said data link (file path) embedded in said dynamic API. As shown in the Program Listing Deposit, at each data object instantiation, a database handler called entrance is constructed. For example,

[0040] entrance = Manager.getEntrance(PrimaryHolders.dataDir, ssn);

[0041] This is to say, the parent directory of a PrimaryHolder data file is PrimaryHolders. The returned object entrance points to the instance of specific ssn.

[0042] Print out all ssn, bank account numbers, and balances. *Application D* in the Program Listing Deposit executes this query.

SECURITY

[0043] The present invention delivers access control to each instance of the data. Each XML file has a username element to hold the user credential against which data access can be checked in real time. As shown in said sample applications, user name is a required argument for object instantiation.

[0044] The present invention delivers data encryption to the attribute/field level. For example, ssn - social security number of the primary holder is sensitive data and needs to be encrypted. As ssn is an argument for the PrimaryHolder constructor, an encryption utility is called from the constructor as listed in *API E* in the Program Listing Deposit.

[0045] After setting values for userName, firstName, lastName, and accountNumber, the XML data content *466* looks as follows. You can see ssn gets encrypted.

[0046] <?xml version="1.0"encoding="utf-8" ?>

```
<PrimaryHolder> <ssn>*(^Re#%[KrP$</ssn>
<userName>a_user</userName>
<firstName>John</firstName>
<lastName>Smith</lastName>
<accountNumber>456</accountNumber>
<accountNumber>789</accountNumber>
</PrimaryHolder>
```

COMPARISONS

[0047] Given said bank account example, the present invention differentiates itself throughout the design process, API and persistence layers.

[0048] Relational databases starts with entity relationship diagrams, but most likely with no entity inheritance like bank accounts *380*, *390*, and *396*. Also, relational database systems do not support native arrays or lists.

[0049] From a relational perspective, NXOD creates three tables: *PrimaryHolder* – ssn, firstName, lastName, accountNumbers; *CheckingAccount* – accountNumber, bankName, balance, overdraft; *BrokerageAccount* – accountNumber, bankName, balance, margin.

[0050] A relational database, however, takes a more fragmented approach. Four tables are created: *PrimaryHolder* – ssn, firstName, lastName; *BankAccount* – accountNumber, bankName, balance, ssn; *CheckingAccount* - accountNumber, overdraft; *BrokerageAccount* - accountNumber, margin.

[0051] Now, PrimaryHolder and account tables are linked by ssn instead of accountNumber in NXOD. The obvious fact that the primary holder has several bank accounts becomes hidden among the relationships. Given social security number '23456789' print out his/her bank account numbers. A relational database application will run a SQL statement with externally configured access control:

[0052] SELECT ACCOUNTNUMBER FROM BANKACCOUNTS WHERE SSN=123456789

[0053] The present invention executes in said sample Java application following statements with a built-in security:

[0054] PrimaryHolder holder = new PrimaryHolder(credential, ssn);

[0055] String []accounts = holder.listAccounts();

[0056] An object database would run following statements with pre-manufactured API and an externally configured security:

[0057] IClass iClass = findDataClass(path_to_PrimaryHolder_class);

[0058] IObject object = iClass.constructObject(ssn);

[0059] String []accounts = (String [])object.getPropertyValue(accountNumbers);

[0060] An XML database application would send a bulky SOAP message with externally configured security:

[0061] <?xml version="1.0"encoding="UTF-8" ?> <soapenv:Envelope xmlns:soapenv="ttp://schemas.xmlsoap.org/soap/envelo pe/ "xmlns:xsd="ttp://www.w3.org/2001/XMLSchema"xmlns: xsi="ttp://www.w3.org/2001/XMLSchema-instance" < soapenv:Body> <listAccounts xmlns="rn:PrimaryHolder.gmorpher.com" </listAccounts> </soapenv:Body>

[0062] In the persistence layer, relational, object, and some XML oriented databases store data in one blob file, which makes database vulnerable for data corruptions. Existing native XML databases may store data in a tree of XML files, but each XML has nested structures of data. The database is still vulnerable for local data corruptions.

[0063] The present invention maps the structure of data to file system paths. Each XML data file is flat and holds name/ value pairs, but no nested structures. Each XML data file represents one instance of data, which is a row/tuple from a relational perspective. Therefore, data corruption is quarantined and minimized to the row /tuple level.

[0064] On the performance side, as structure of data is mapped to file system paths, locating a piece of data triggers OS system calls, which are faster than application level method invocations. NXOD has the ability to load any desired row of data on the fly without engaging unrelated data while existing databases need to load the whole blob file or deeply nested XML files into memory even if only a small portion of data is actually accessed. Therefore, NXOD has smaller memory footprints and faster transactions.

Database Core

[0065] The present invention provides means and steps for building a processing center to map Get/Set operations in the API layer to XML content changes in the persistence layer. Get operations in the API comprising of getxxx() and listxxx() where xxx is the data field name, are for data retrieval. Set operations in the API comprising of setxxx(), deletexxx(), addxxx(), removexxx(), and commit(), are for data modifications. addxxx() and removexxx() are for a list of values/references.

[0066] *Fig. 5* can be viewed as an expansion of *Fig. 2* to drill down to the database core which comprises of four major components: *560, 562, 564,* and *568.*

[0067] Dynamic API *540* starts a representative get operation getBalance(). Core Entrance *560* translates it into getDouble("balance"); then Core Porter *562* into get("balance") The computer-readable program code of the present invention utilizes Apache Xerces XML parser for component *568*, which translates the get operation further into getNodeValue() and fetches data from XML Files *580*.

[0068] API *540* also starts a representative set operation setBalance(). Core Entrance *560* translates it into setDouble(). Core Porter *562* sets the value to Core Cache *564*. To persist cumulative set operations, the external application calls commit() exposed via API *540*, which saves the changes to XML Files *580* and cleans up Core Cache *564*.

[0069] Create, update, delete are three major NXOD operations at the data field level. The set operation in dynamic API *540* causes a new field value to be created if it is not preexistent. Otherwise, it is an update operation to overwrite existent data. Therefore, create and update map to setxxx() in dynamic API *540* for a single value/reference, to addxxx() for a list of values/references. And delete maps to deletexxx() or removexxx(). See the Program Listing Deposit for Java code examples.

[0070] At the instance/tuple level, NXOD starts with the instantiation of a data object. If the instance does not exist, it is an insertion operation; update operation, otherwise. Delete is accomplished by removing the correspondent XML file.

PROGRAM LISTING DEPOSIT

[0071] [/*** * LISTED BELOW IS JAVA CODE DISCLOSURE. * JAVA EXCEPTION HANDLING CODE IS STRIPPED OFF FOR SIMPLICITY. **//************************* Application A **************************/PrimaryHolder holder = new PrimaryHolder("a_user", "123"); holder.setFirstName("a_firstName"); holder.setLastName("a_lastName"); holder.addAccount("456"); holder.addAccount("789"); Porter.commit();/************************** Application B ****************************/CheckingAccount cAccount = new CheckingAccount("a_user", "456"); cAccount.setBankName("a_bank"); cAccount.setBanlance(new Double("2000.68")); cAccount.setOverdraft(new Dou-

```
ble("1000.00"));
Porter.commit();/************************** Application C
***************************/ BrokerageAccount bAccount
= new BrokerageAccount("a_user", "789"); bAc-
count.setBankName("b_bank"); bAccount.setBalance(new
Double("8000.26")); bAccount.setMargin(new
Boolean(true)); Porter.commit();/**************************
Application D ***************************/// starts with
the query class PrimaryHolders String []holders = Primary-
Holders.listPrimaryHolders(); int hc = holders.length;
for(int i = 0; i < hc; i++) { String ssn = holders[i]; // ad-
min privilege is needed to view all records PrimaryHolder
holder = new PrimaryHolder("admin", ssn); String
[]accounts = holder.listAccounts(); int ac = ac-
counts.length; for(int j = 0; j < ac; j++) { String account-
Number = accounts[j]; // Each account number points to
a BankAccount class. BankAccount account = null;
if(BankAccounts.isCheckingAccount(accountNumber)) ac-
count = new CheckingAccount(credential, accountNum-
ber); el-
seif(BankAccounts.isBrokerageAccount(accountNumber))
account = new BrokerageAccount(credential, account-
Number); System.out.println(ssn + " " + accountNumber +
" US$" + account.getBalance()); }
}/************************* API E
**************************/public PrimaryHolder(String
userName, String ssn) { this.userName = userName;
this.ssn = ssn; set(ssn, encrypt(ssn));
}/*************************API and Database
Core**************************//************************
*Claims 10, 11**************************/public class
PrimaryHolders{ public static File dataDir = new
File(parent, "directoryNameForPrimaryholders")
dataDir.mkdirs(); public String[] listPrimaryHolders() { re-
turn Manager.list(dataDir); }}public interface IIdentity{
public String getUserName();}public interface IPrimary-
Holder extends IIdentity{ public String getSocialSecuri-
tyNumber(); public String getFirstName(); public void set-
FirstName(String firstName); public void deleteFirst-
Name(); public String getLastName(); public void setLast-
Name(String lastName); public void deleteFirstName();
public String[] listAccounts(); public void addAc-
count(String accountNumber); public void removeAc-
count(String accountNumber);}public final class Primary-
Holder implements IPrimaryHolder{ private Entrance en-
trance; public PrimaryHolder(String userName, String ssn)
```

```
{ entrance = Manager.getEntrance(PrimaryHolders.dataDir,
ssn); entrance.set("userName", userName); en-
trance.set("ssn", ssn); } public String getUserName() { re-
turn entrance.get("userName"); } public String getSocialSe-
curityNumber() { return entrance.get("ssn"); } public String
getFirstName() { return entrance.get("firstName"); } public
void setFirstName(String firstName) { en-
trance.set("firstName", firstName); } public void delete-
FirstName() { entrance.delete("firstName"); } public String
getLastName() { return entrance.get("lastName"); } public
void setLastName(String lastName) { en-
trance.set("lastName", lastName); } public void deleteLast-
Name() { entrance.delete("lastName"); } public String[] lis-
tAccounts() { return entrance.list("accountNumber"); }
public void addAccount(String accountNumber) { en-
trance.add("accountNumber", accountNumber); } public
void removeAccount(String accountNumber) { en-
trance.remove("accountNumber", accountNumber);
}}public class BankAccounts{ public static File dataDir =
new File(parent, "directoryNameForBankaccounts")
dataDir.mkdirs(); public boolean isCheckingAccount(String
accountNumber) { return Man-
ager.having(CheckingAccounts.dataDir, accountNumber); }
```

```
public boolean isBrokerageAccount(String accountNum-
ber) { return Manager.having(BrokerageAccounts.dataDir,
accountNumber); }}public class CheckingAccounts extends
BankAccounts{ public static File dataDir = new File(parent,
"directoryNameForCheckingaccounts") dataDir.mkdirs();
public String[] listCheckingAccounts() { return Man-
ager.list(dataDir); }}public class BrokerageAccounts ex-
tends BankAccounts{ public static File dataDir = new
File(parent, "directoryNameForBrokerageaccounts")
dataDir.mkdirs(); public static String[] listBrokerageAc-
counts() { return Manager.list(dataDir); }}public interface
IBankAccount extends IIdentity{ public String getAccount-
Number(); public String getBankName(); public void set-
BankName(String bankName); public void delete-
BankName(); public Double getBalance(); public void set-
Balance(Double balance); public void deleteBal-
ance();}public interface ICheckingAccount extends
IBankAccount{ public Double getOverdraft(); public void
setOverdraft(Double overDraft); public void deleteOver-
draft();}public interface IBrokerageAccount{ public Boolean
getMargin(); public void setMargin(Boolean margin); public
void deleteMargin();}public abstract class BankAccount
implements IBankAccount{ protected Entrance entrance;
```

```
public String getUserName() { return en-
trance.get("userName"); } public String getAccountNum-
ber() { return entrance.get("accountNumber"); } public
String getBankName() { return entrance.get("bankName"); }
public void setBankName(String bankName) { en-
trance.set("bankName", bankName); } public String delete-
BankName() { return entrance.delete("bankName"); } public
Double getBalance() { return en-
trance.getDouble("balance"); } public void setBal-
ance(Double balance) { entrance.setDouble("balance", bal-
ance); } public Double deleteBalance() { return en-
trance.delete("balance"); }}public class CheckingAccount
extends BankAccount implements ICheckingAccount{
public CheckingAccount(String userName, String account-
Number) { entrance = Man-
ager.getEntrance(CheckingAccounts.dataDir, account-
Number); entrance.set("userName", userName); en-
trance.set("accountNumber", accountNumber); } public
Double getOverdraft() { return en-
trance.getDouble("overDraft"); } public void setOver-
draft(Double overDraft) { entrance.setDouble("overDraft",
overDraft); } public Double deleteOverdraft() { return en-
trance.delete("overDraft"); }}public class BrokerageAccount
```

```
extends BankAccount implements IBrokerageAccount{
public BrokerageAccount(String userName, String ac-
countNumber) { entrance = Man-
ager.getEntrance(BrokerageAccounts.dataDir, account-
Number); entrance.set("userName", userName); en-
trance.set("accountNumber", accountNumber); } public
Boolean getMargin() { return en-
trance.getBoolean("margin"); } public void setMar-
gin(Boolean margin) { entrance.setBoolean("margin", mar-
gin); } public Boolean deleteMargin() { return en-
trance.delete("margin");
}}/**************************Claims 10,
12**************************/public class Entrance{ pri-
vate Porter porter; public Entrance(File dataFile) { porter =
new Porter(dataFile); } public String get(String name) { re-
turn porter.get(name); } public void set(String name, String
value) { porter.set(name, value); } public Boolean get-
Boolean(String name) { String value = porter.get(name);
return new Boolean(value); } public void setBoolean(String
name, Boolean value) { porter.set(name, value.toString()); }
public Double getDouble(String name) { String value =
porter.get(name); return new Double(value); } public void
setDouble(String name, Double value) { porter.set(name,
value.toString()); } public String[] getList(String name) { re-
turn porter.getList(name); } public void add(String name,
String value) { porter.add(name, value); } public void re-
move(String name, String value) { porter.remove(name,
value); } public void delete(String name) {
porter.delete(name); }}public class Porter{ public static
volatile Hashtable commitTable = new Hashtable(); public
static volatile Hashtable removalTable = new Hashtable();
private File dataFile; public Porter(File dataFile) {
this.dataFile = dataFile; } public static void commit() {
Enumeration files = commitTable.keys(); Enumeration re-
movalFiles = removalTable.keys();
while(files.hasMoreElements())
save((File)files.nextElement(), false);
while(removalFiles.hasMoreElements())
save((File)removalFiles.nextElement(), true); } public static
void rollback() { commitTable.clear(); removalTable.clear();
} public String get(String name) { String value; Hashtable
fields = (Hashtable)commitTable.get(dataFile); if(fields ==
null) value = getFromStorage(name); else { value =
(String)fields.get(name); if(value == null) value = get-
FromStorage(name); } return value; } public void set(String
name, String value) { set(name, value, false); } private void
```

```
set(String name, String value, boolean isList) { Hashtable
fields = (Hashtable)commitTable.get(dataFile); if(!isList)
fields.put(name, value); else { Set set =
(Set)fields.get(name); set.add(value); fields.put(name, set);
} } private String getFromStorage(String name) { Document
document = getDocument(dataFile); Element element =
document.getElementsByTagName(name).item(0); Text
text = (Text)element.getFirstChild(); return
text.getNodeValue(); } private static void save(File file,
boolean isRemoval) { Hashtable fields = null; if(isRemoval)
fields = (Hashtable)removalTable.get(file); else fields =
(Hashtable)commitTable.get(file); save(file, fields, isRe-
moval); } private static void save(File file, Hashtable fields,
boolean isRemoval) { Document document = getDocu-
ment(file); Enumeration keys = fields.keys();
while(keys.hasMoreElements()) { String name =
(String)keys.nextElement(); Object object =
fields.get(name); if(object instanceof Set) { if(isRemoval)
saveRemoval(name, (Set)object, document); else
add(name, (Set)object, document); } else saveSingle(name,
(String)object, document); } write(file, document); } private
static void add(String name, Set set, Document document)
{ Element rootElement = docu-
ment.getDocumentElement(); Set currentSet =
getList(name, document); Iterator iterator = set.iterator();
while(iterator.hasNext()) { Object value = iterator.next();
if(!currentSet.contains(value)) { Element element = docu-
ment.createElement(name); rootEle-
ment.appendChild(element); Text text = docu-
ment.createTextNode((String)value); ele-
ment.appendChild(text); } } } private static Set
getList(String name, Document document) { Set set = new
TreeSet(); NodeList list = docu-
ment.getElementsByTagName(name); for(int i = 0; i <
list.getLength()); i++)
set.add(list.item(i).getFirstChild().getNodeValue()); return
set; } private static void saveSingle(String name, String
value, Document document) { Element element = docu-
ment.getElementsByTagName(name).item(0); Element
rootElement = document.getDocumentElement(); rootEle-
ment.appendChild(element); Text text =
(Text)element.getFirstChild(); if(text == null) { text = doc-
ument.createTextNode(value); element.appendChild(text);
} else text.setNodeValue(value); } private static void
write(File file, Document document) { OutputFormat for-
mat = new OutputFormat(document, utf-8, true);
```

```
FileWriter writer = new FileWriter(file); XMLSerializer seri-
alizer = new XMLSerializer(writer, format); serial-
izer.asDOMSerializer(); serializer.serialize(document);
writer.close(); } private static Document getDocument(File
file) { DocumentBuilderFactory dbf = DocumentBuilder-
Factory.newInstance(); DocumentBuilder builder =
dbf.newDocumentBuilder(); return builder.parse(file); }
public Set getList(String name) { Document document =
getDocument(dataFile); return getList(name, document); }
public void add(String name, String value) { set(name,
value, true); } public void remove(String name, String
value) { Hashtable fields =
(Hashtable)removalTable.get(dataFile); Set set =
(Set)fields.get(name); set.remove(value); } private static
void saveRemoval(String name, Set set, Document docu-
ment) { NodeList list = docu-
ment.getElementsByTagName(name); Vector vector = new
Vector(); int count = list.getLength(); for(int i = 0; i <
count; i++) { Node node = list.item(i); Text text =
(Text)node.getFirstChild(); if(text != null) { String value =
text.getNodeValue(); if(set.contains(value)) vec-
tor.add(node); } } Element rootElement = docu-
ment.getDocumentElement(); int size = vector.size();
```

```
for(int j = 0; j < size; j++) rootEle-
ment.removeChild((Node)vector.elementAt(j)); } public
void delete(String name) { porter.set(name, ""); }}public
class Manager{ public static Entrance getEntrance(File par-
ent, String name) { File dataFile = new File(parent, name +
".xml"); dataFile.createNewFile(); return new En-
trance(dataFile); } public static final boolean having(File
dataDir, String key) { String []list = dataDir.list(); int size =
list.length; for(int i = 0; i < size; i++) { String fileName =
list[i].trim(); String name = fileName.substring(0, file-
Name.length() - 3);
if(name.trim().compareToIgnoreCase(key) == 0) return
true; } return false; } public static final String[] list(File
dataDir) { if(dataDir == null) return null; String []list =
dataDir.list(); int size = list.length; for(int i = 0; i < size;
i++) { String fileName = list[i]; String name = file-
Name.substring(0, fileName.length() - 3); list[i] = name; }
return list; } }]
```